Sentinel

<Delve into Web Dev>

DEFENDING OUR DIGITAL WAY OF LIFE

Recap

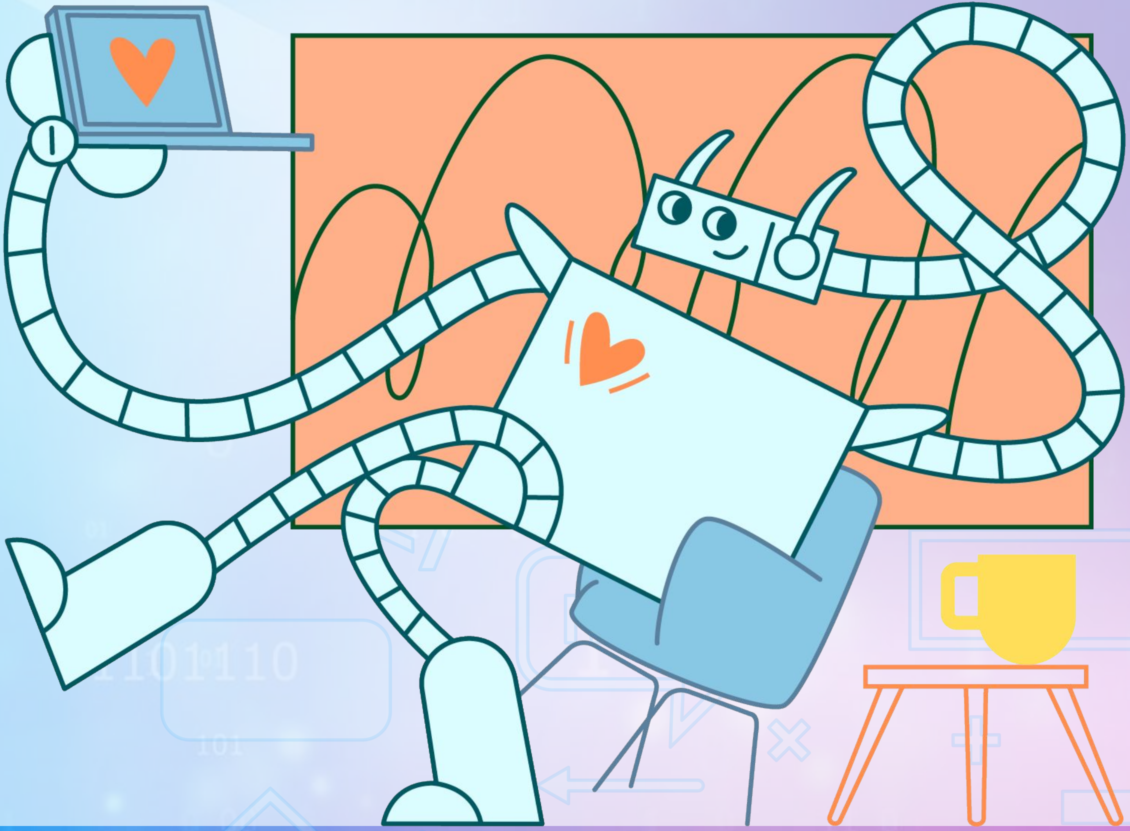
In the previous lesson, we learnt more about string formatting and the various built in methods!

Recap

Newline	`\n`
To represent “ character	`let myString = "\"hello\""`
To get the char at a particular index	`let firstChar = myString[0]`
To get the length of a string	`myString.length`

Recap

Convert string to uppercase	`myString.toUpperCase()`
Convert string to lowercase	`myString.toLowerCase()`
Replace all occurrences of a substring	`myString.replace("money", "tea")`
Get chars between starting and ending indexes	`myString.slice(0, 2)`



Intro to Functions

Learning Objective

How to define a function in JavaScript

How to call a function

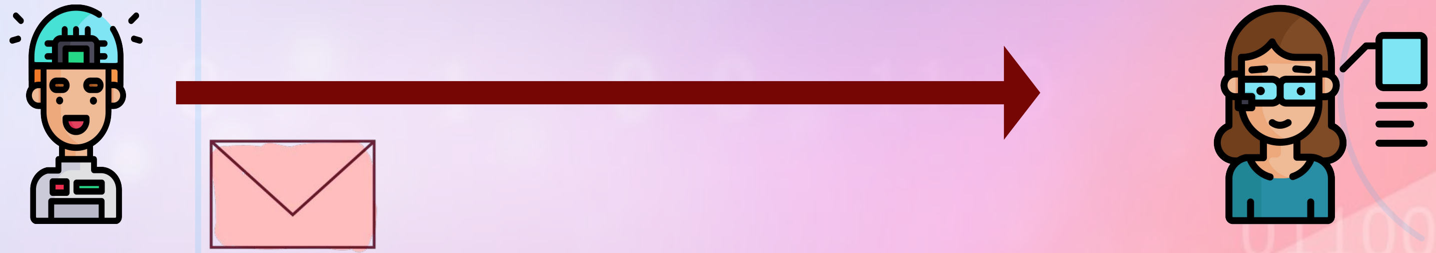
How to call functions within other functions

Differentiate between global and local variables

How to return a value from a function

Activity

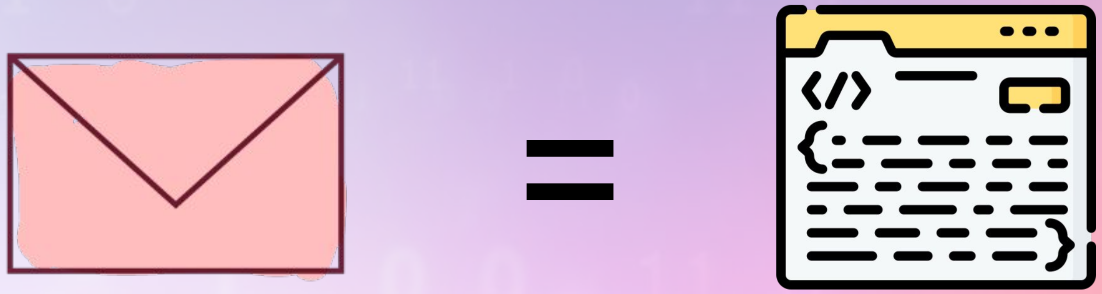
1. Split into groups of 4
2. Each one gets an envelope with a specific task written on it
3. Each member will take turns to perform their task and put their results into their envelope
4. Pass the envelope to the next member



Functions

Can be seen as "containers" for code.

Like the envelopes – they contain the instructions to perform.



Function Definition

To define a function in JavaScript we use the syntax:

The *function* keyword means you want this to be a function

This is the name of the function

```
function functionName() {
    // Function code
}
```

Within the curly braces we put the body of the function

The parameters of the function go inside these parentheses. This function *has no parameters*

Function Call

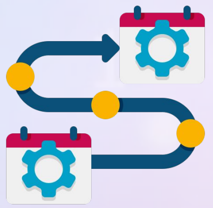
To call a function we use this syntax:

```
functionName()
```

```
function functionName() {
    // Function code
}
```

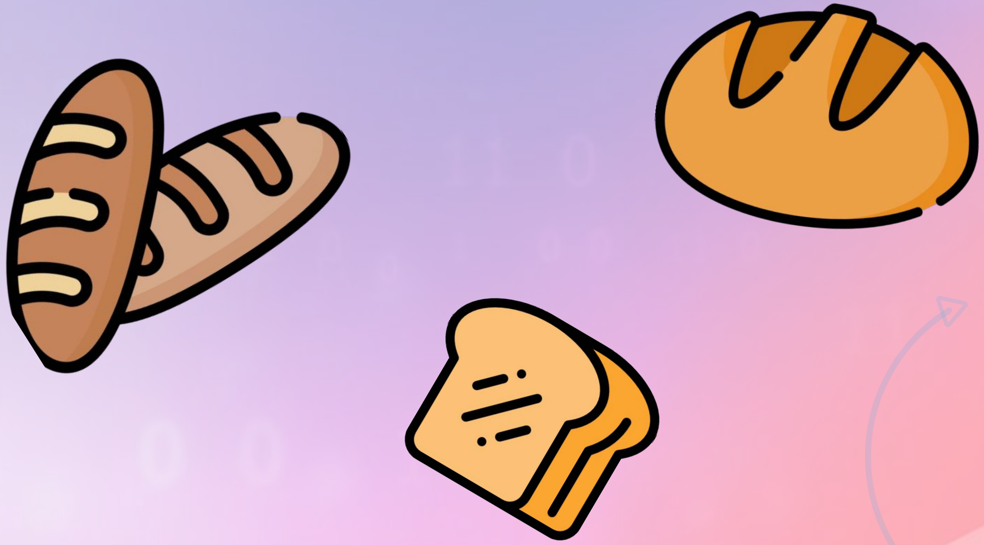
Let's see how functions change the flow of execution:

JStutor



Bake Bread

Take a few minutes to complete the first "functions" exercise – Bake Bread



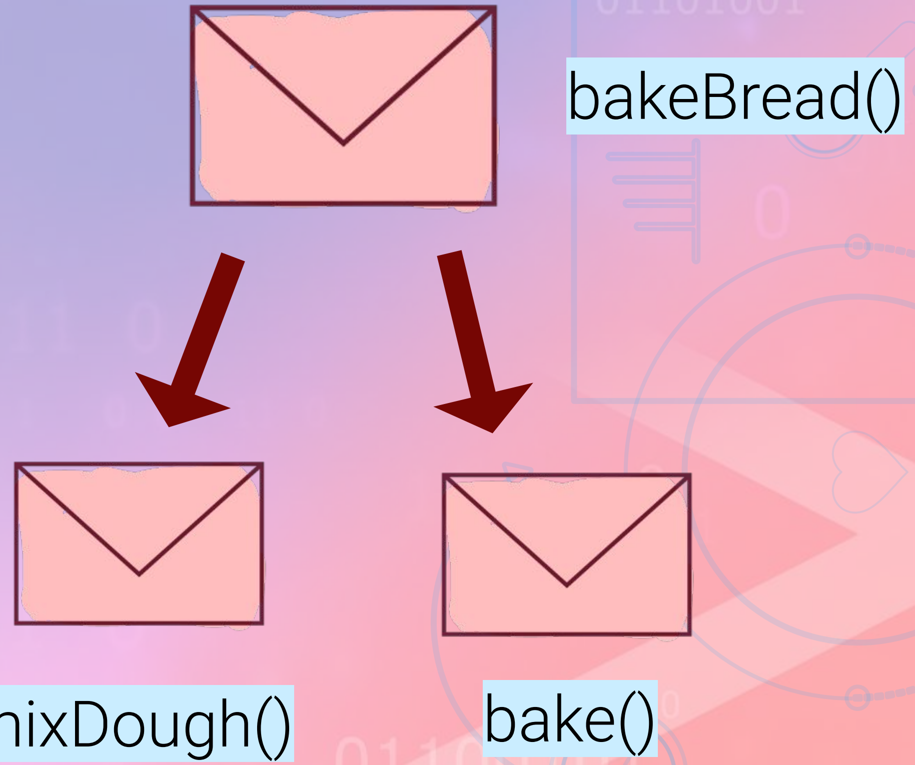
Solution

```
function mixDough() {
    console.log("The dough has been mixed")
}

function bake() {
    console.log("The bread has been baked!")
}

mixDough()
bake()
```

Calling Functions from Functions

We can also call functions from within other functions

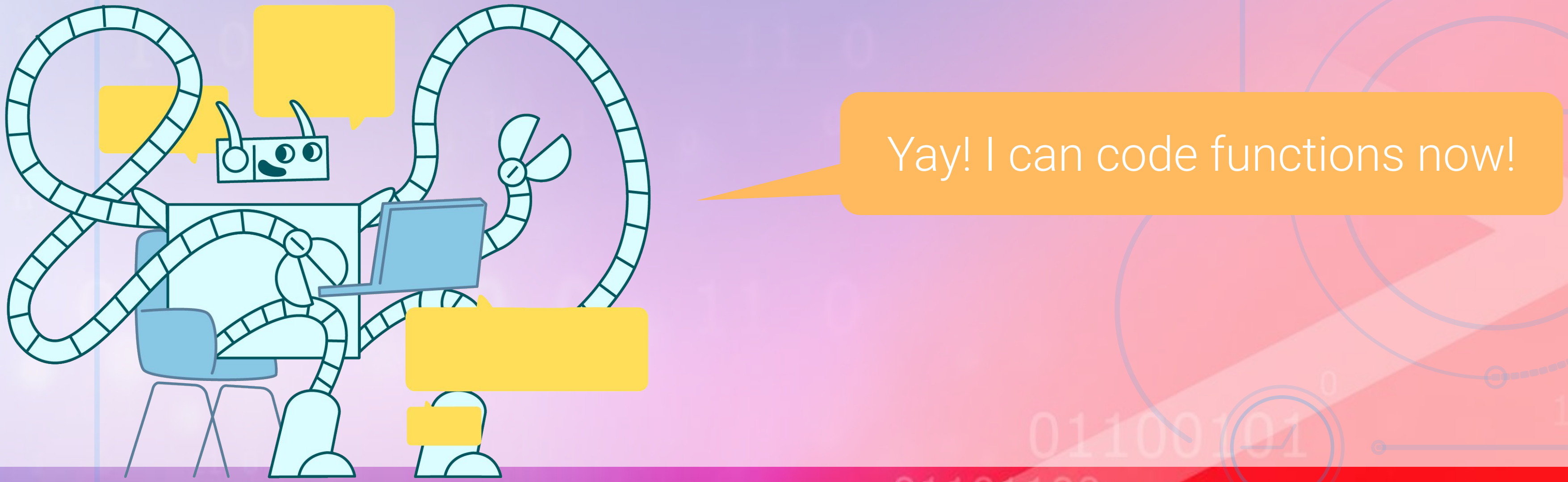
```
function bakeBread() {
    mixDough()
    bake()
}

bakeBread()
```



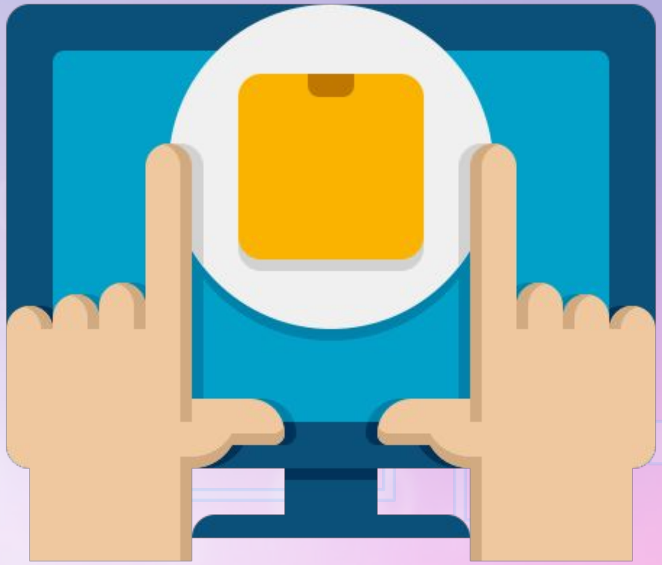
More on that later…

We'll talk more about functions at a later stage.

For now, whenever you see exercises that specify to "write a function that…" – you'll know what to do ☺



Scopes



let

Remember how we assign variables as var?

```
var a = 3
```

From now on we will use **let**

```
let a = 3
```

To learn more about the difference between var and let: https://www.geeksforgeeks.org/difference-between-var-and-let-in-javascript/

const

There is also the **const** declaration

```
const a = 3
```

This creates a constant where the value cannot be changed through reassignment

```
const a = 3
a = 4
```

Variables in functions

```
function doA() {
    let a = 3
}

console.log(a)  // What will be printed?
```

Concept: Local Variables

```
function doA() {
    let a = 3
}
```

doA Locals:

Name	Value
a	3

This is called the “frame” of the function

Concept: Global Variables

```
function doA() {
    let a = 3
    console.log(a)
}

let a = 1337

doA()
console.log(a)
```

doA Locals:

Name	Value
a	3

Global
s:

Name	Value
a	1337

Concept: Global Variables

```
let a = 1337

function doA() {
    a = 3
    console.log(a)
}

doA()
console.log(a)
```

doA Locals:

Name	Value

Globals:

Name	Value
a	1337

Concept: Global Variables

```
function doA() {
    a = 3
    console.log(a)
}

doA()
```

doA Locals:

Name	Value

Globals:

Name	Value
a	3

Concept: Block Scope

```
let a = 3
console.log(a)

if (a == 3) {
    let a = 1337
    console.log(a)
}

console.log(a)
```

Globals:

Name	Value
a	3
a (block 1)	1337

Concept: Block Scope

```
var a = 3
console.log(a)

if (a == 3) {
    var a = 1337
    console.log(a)
}

console.log(a)
```

Global

Name	Value
a	1337

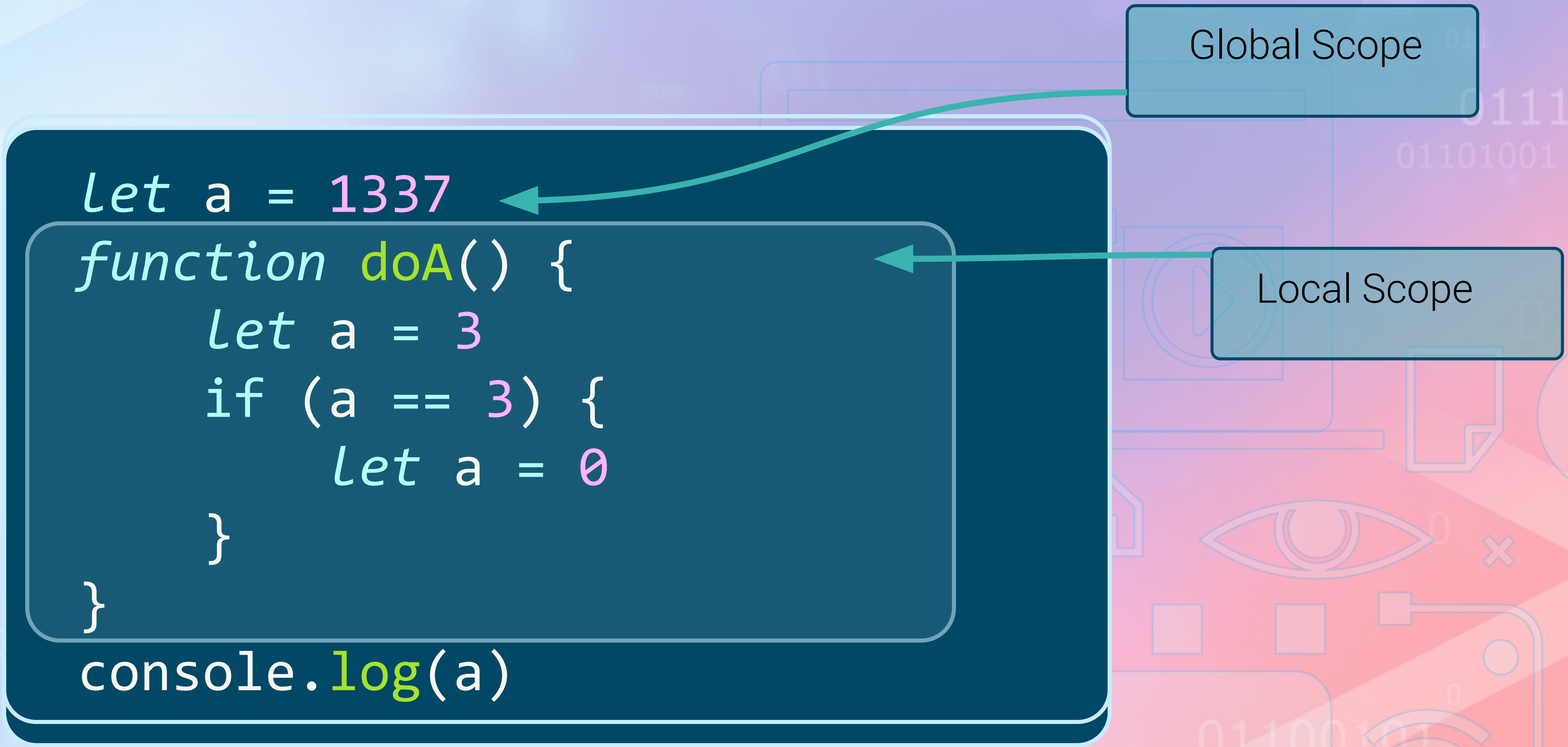
let

That’s why we use let!

It protects us from accidentally modifying local or global variables

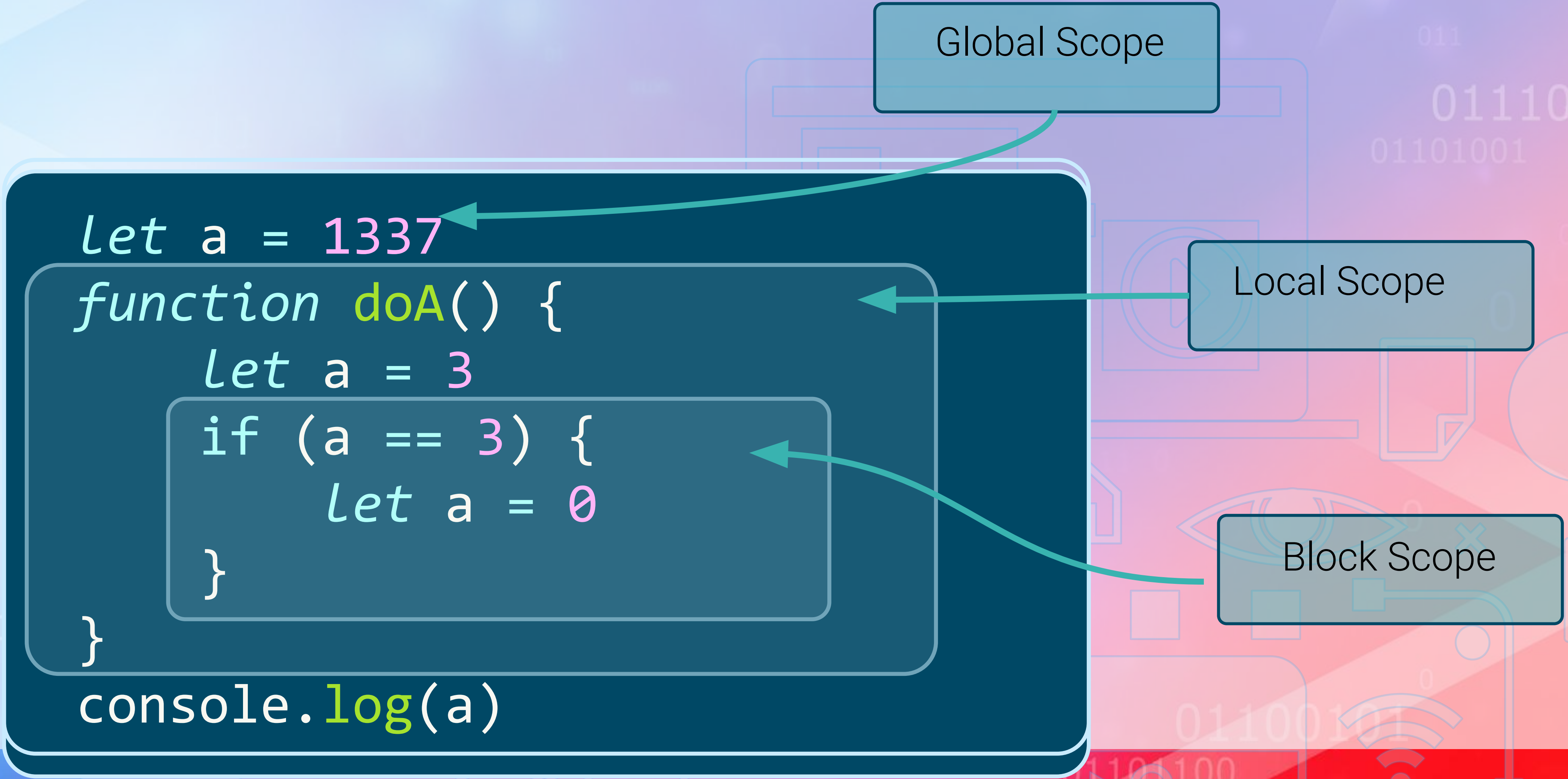
```
let a = 3
```

Summary: Scopes



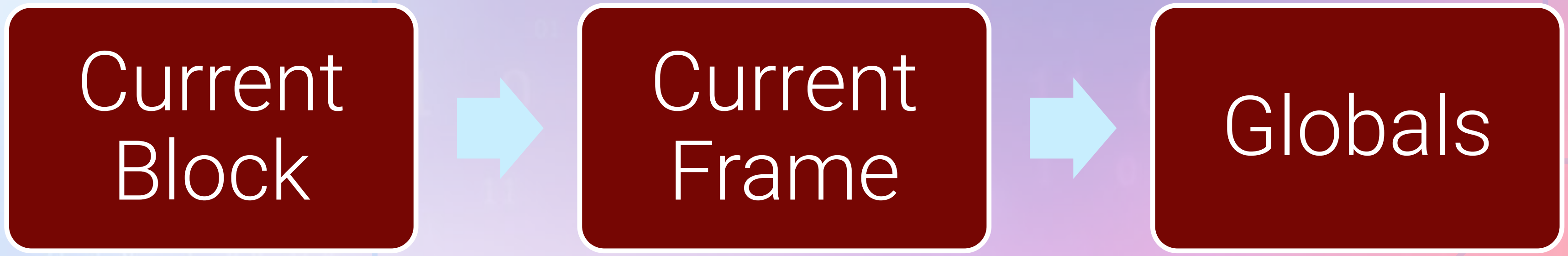
```
let a = 1337
function doA() {
    let a = 3
    if (a == 3) {
        let a = 0
    }
}
console.log(a)
```

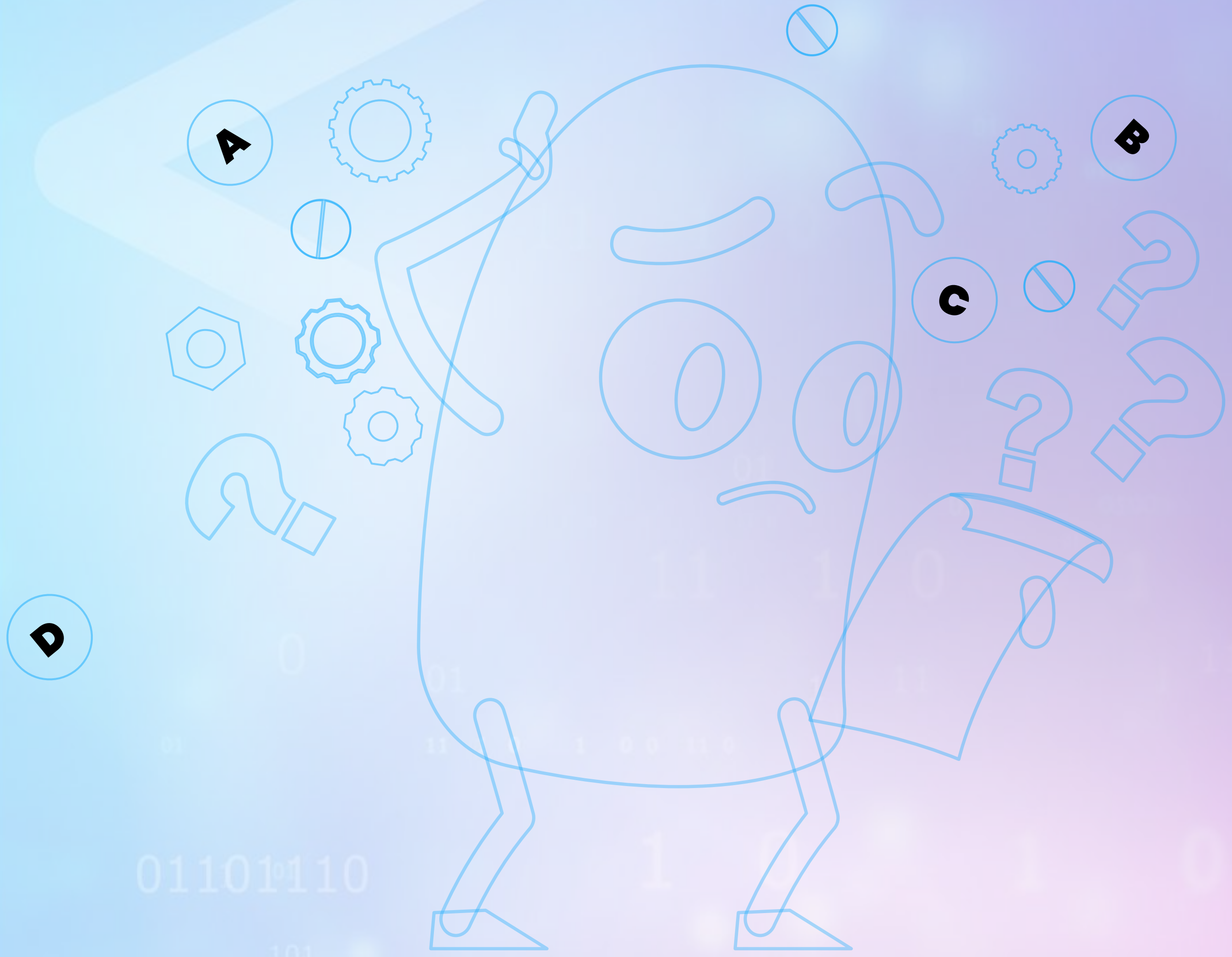
Summary: Scopes



```
let a = 1337
function doA() {
    let a = 3
    if (a == 3) {
        let a = 0
    }
}
console.log(a)
```

Variables Search



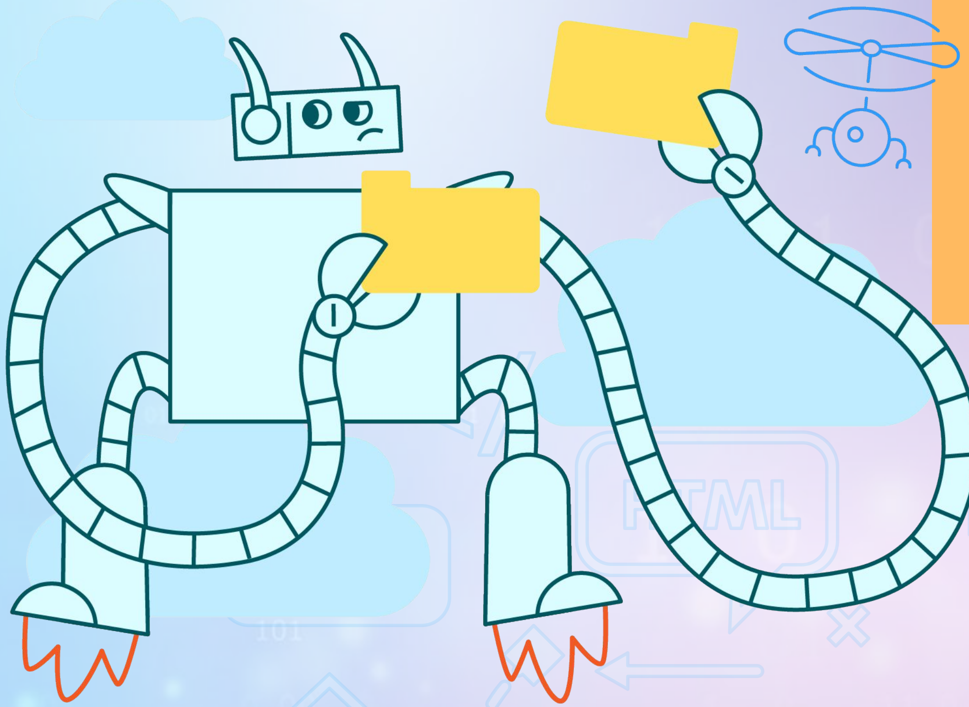


Questions?

Sentinel

Your Turn!

> Play around, have fun, ask questions!



Functions Cont’d

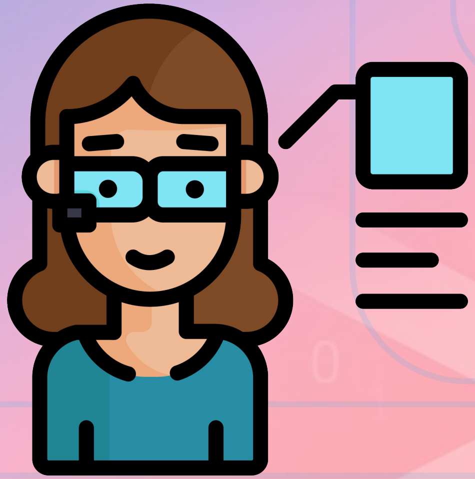
Recap

To declare a function

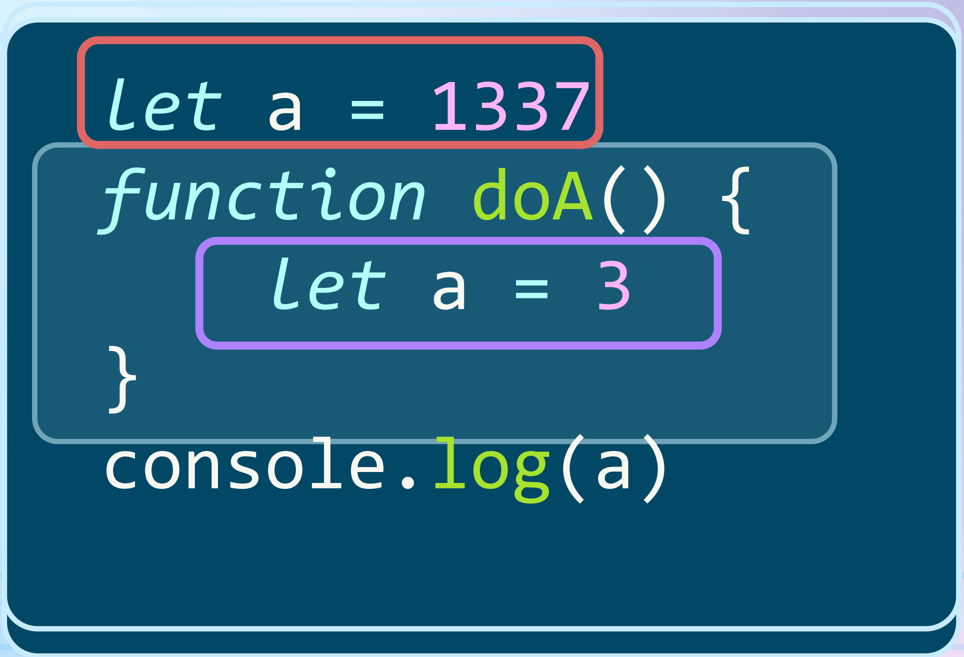
```
function functionName() {
    // Function code
}
```

To call the function

```
functionName()
```



Recap



```
let a = 1337
function doA() {
    let a = 3
}
console.log(a)
```

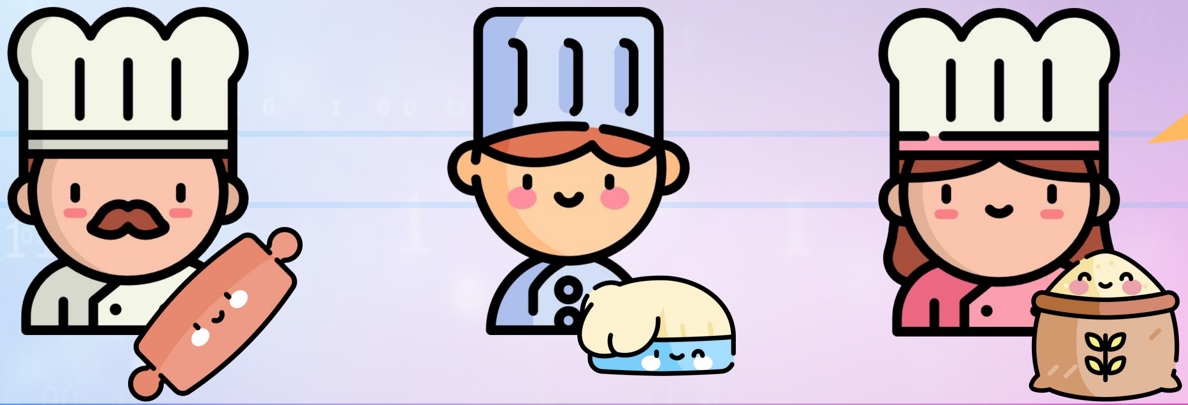
Local Variable: When you define a variable in a function

Global Variable: When you define a variable outside a function

Generic Recipe

Not all bread dough is the same

There are many different recipes

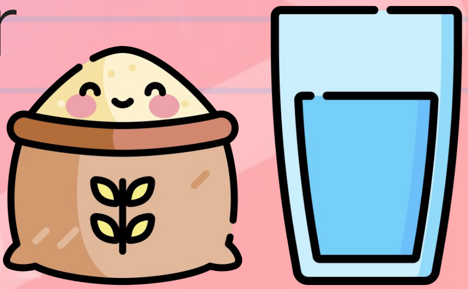

Does that mean we need a different baker for each type of bread?

Generic Recipe

What does the baker need to know?

The amount of each ingredient in the dough!

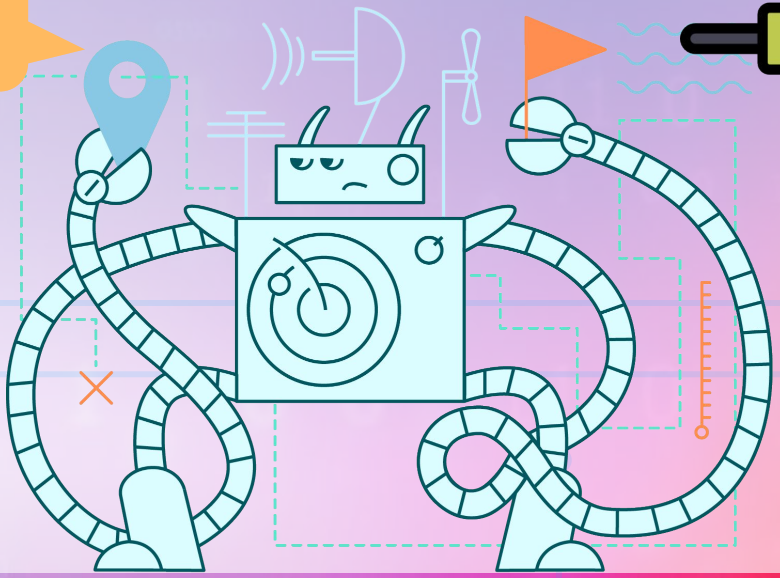
For now we'll simplify this to flour (fla!) and water

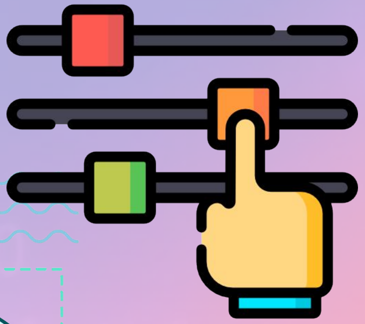


Generic Function

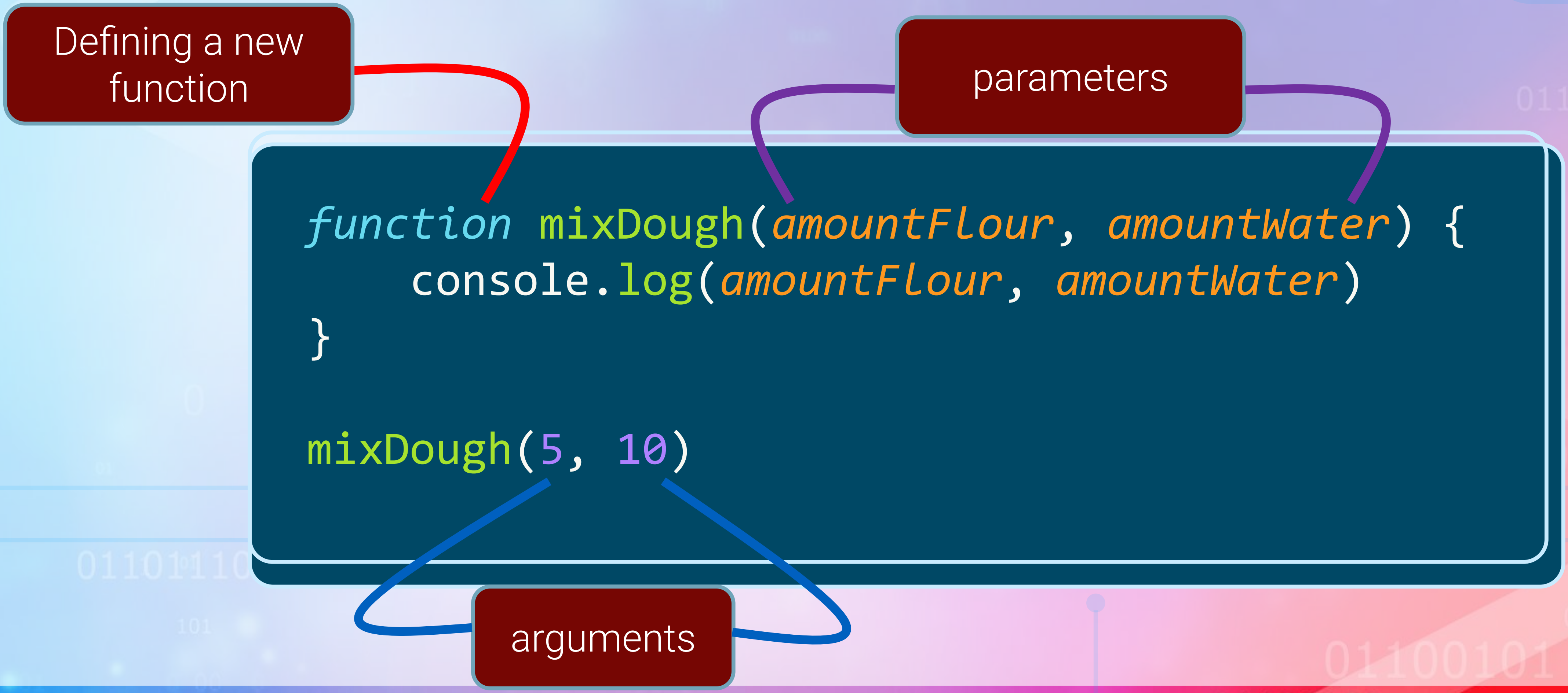
So how could we accommodate our mixDough function to be generic?

Parameters!





Function Parameters



```
function mixDough(amountFlour, amountWater) {
    console.log(amountFlour, amountWater)
}

mixDough(5, 10)
```

Function Parameters

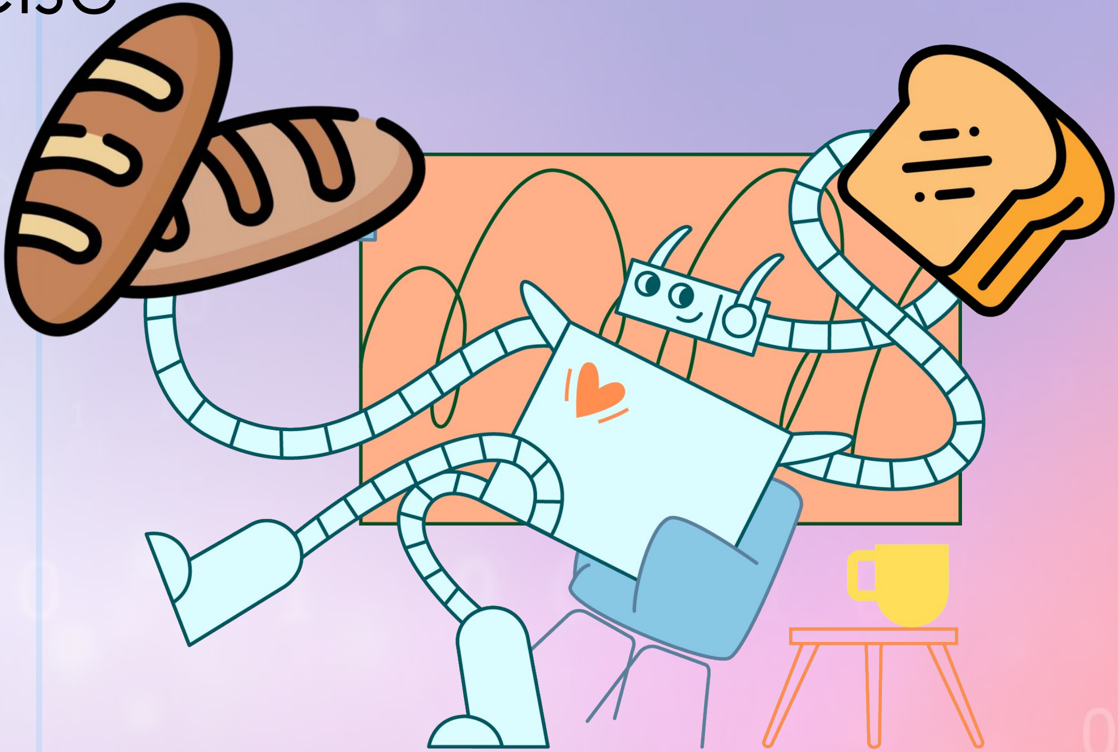
```
function mixDough(amountFlour, amountWater) {
    console.log(amountFlour, amountWater)
}

mixDough(5, 10) // Bread
mixDough(1, 10) // Pancakes
```

mixDough Locals:

Name	Value
amountFlour	1
amountWater	10

Generic Bread

Take a few minutes to complete the “Generic Bread” exercise



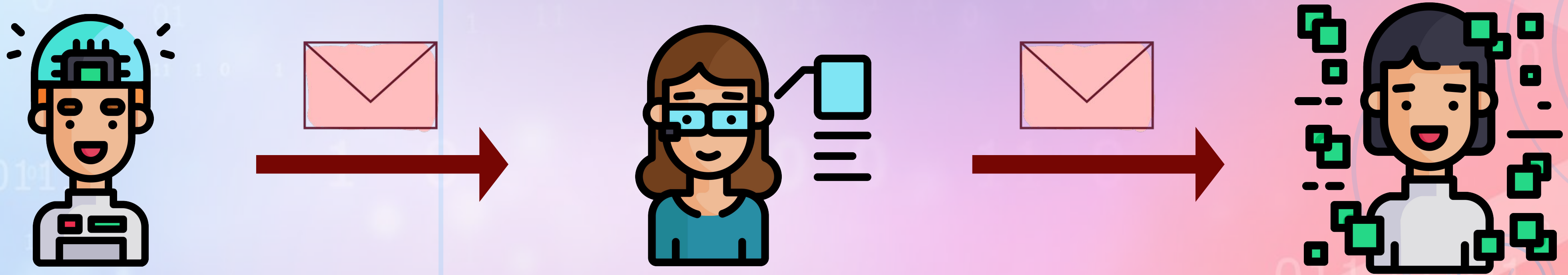
Solution

```
function mixDough(amountFlour,amountWater) {
    console.log(`Mixing ${amountFlour}g flour and
${amountWater}ml water in to dough`)
}

function bake(bakeTime) {
     console.log(`The bread has been baked for
${bakeTime} minutes!`)
}

mixDough(250,50)
bake(25)
```

Envelop Game

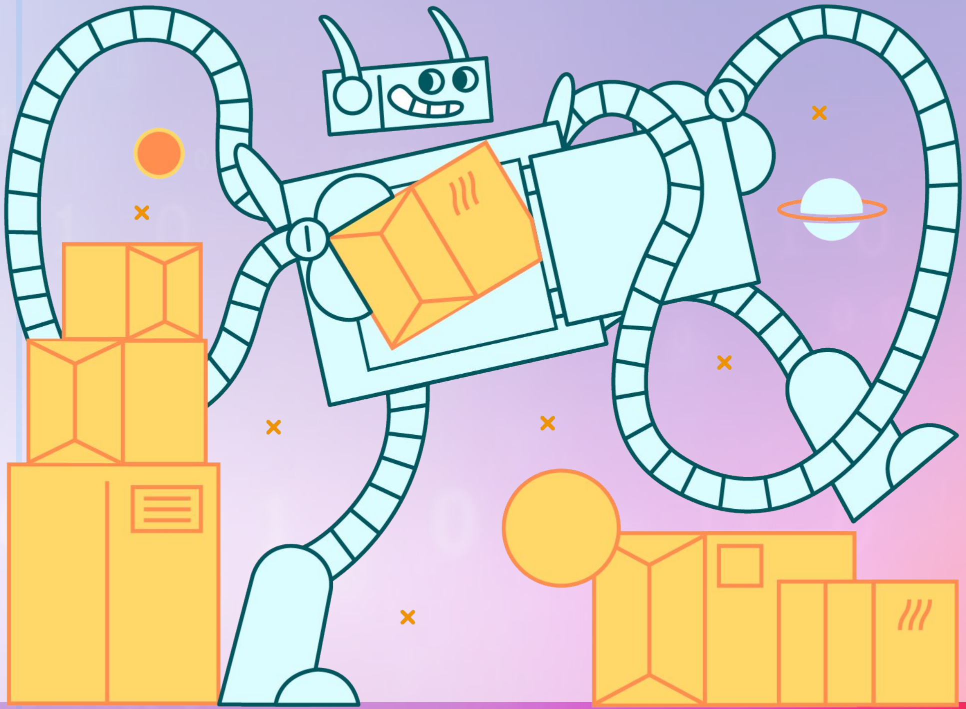
Remember the envelope game?

Each one performed a task, put the result in an envelope and transferred it to the next person



Envelop Game

That way we were able to chain a few small operations in order to complete a more complex task



Return Values

We can "return" a result from a function!

```
function add(lhs, rhs) {
    return lhs + rhs
}

let result = add(1014, 323)
console.log(result)
```

Return the result

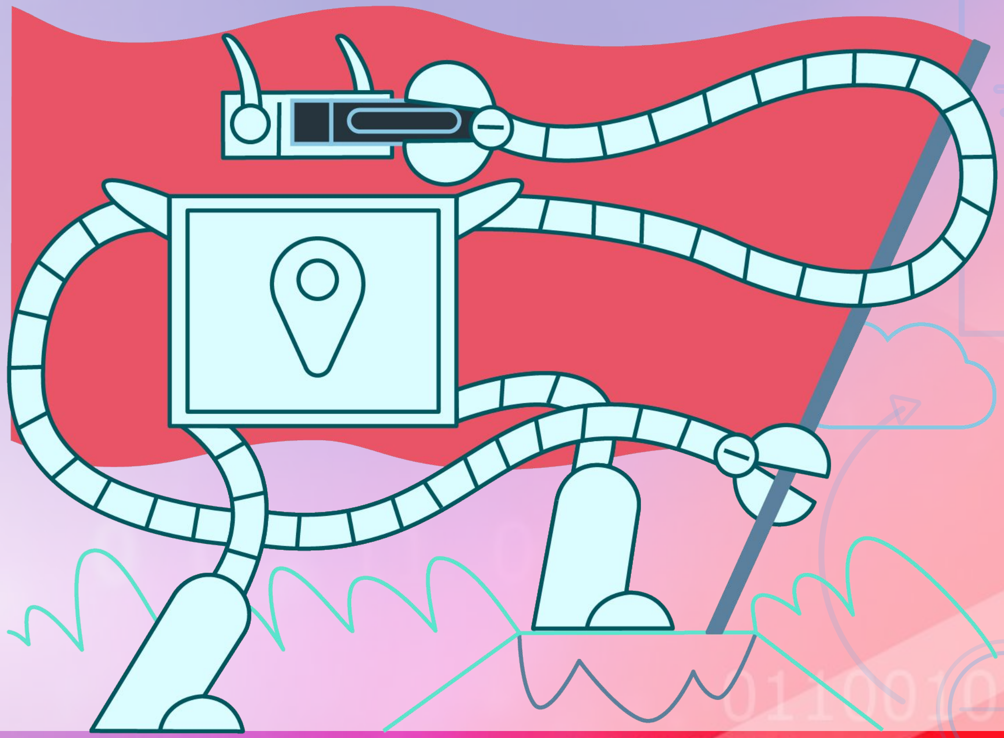
Return Values

Then we can pass this return value to other functions!

```
function add(lhs, rhs) {
    return lhs + rhs
}

alert(String(add(1014, 323)))
```

Return of the Bread

Take a few minutes to complete the “Return of the Bread” exercise



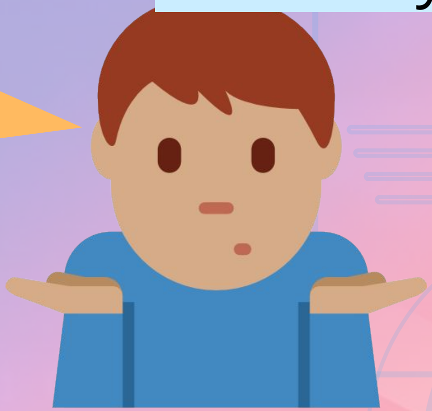
Solution

```javascript
function mixDough(amountFlour,amountWater) {
    console.log(`Mixing ${amountFlour}g flour and
${amountWater}ml water in to dough`)
    return "dough"
}

function bake(bakeTime,dough) {
    console.log(`The bread has been baked for ${bakeTime}
minutes!`)
    if (dough == "dough") {
      return "bread"
    }
}
let dough = (mixDough(250,50))
console.log(bake(25, dough))
```

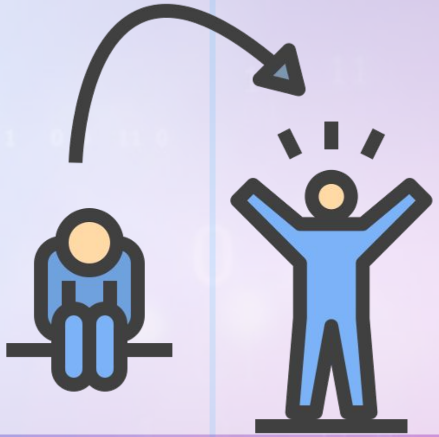
But why bother?

"Ok but I still don't understand what's so good about these function things..."

rolls eyes🙄



Get up.



Abstraction!

Using a single command, we were able to achieve a very very complicated task!

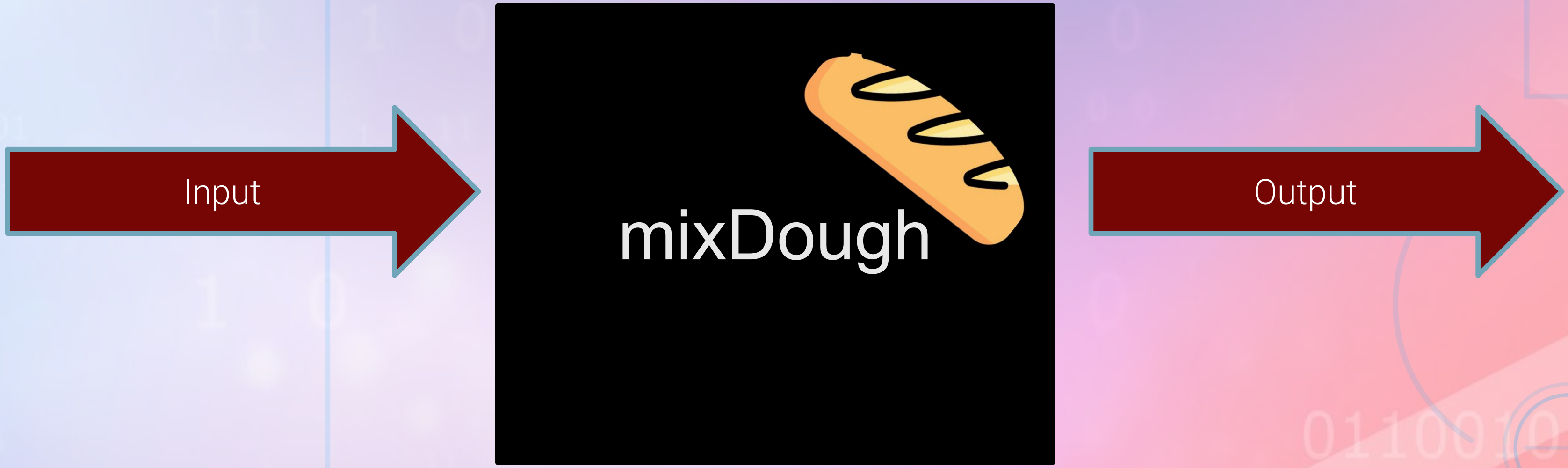
And we don't need to know anything about what happens behind the scenes.

The "implementation details" are hidden

Functions are Abstractions

You don't have to be a baker to call the mixDough function

You just need to know the inputs and outputs



Abstractions Lead To

Code that is more

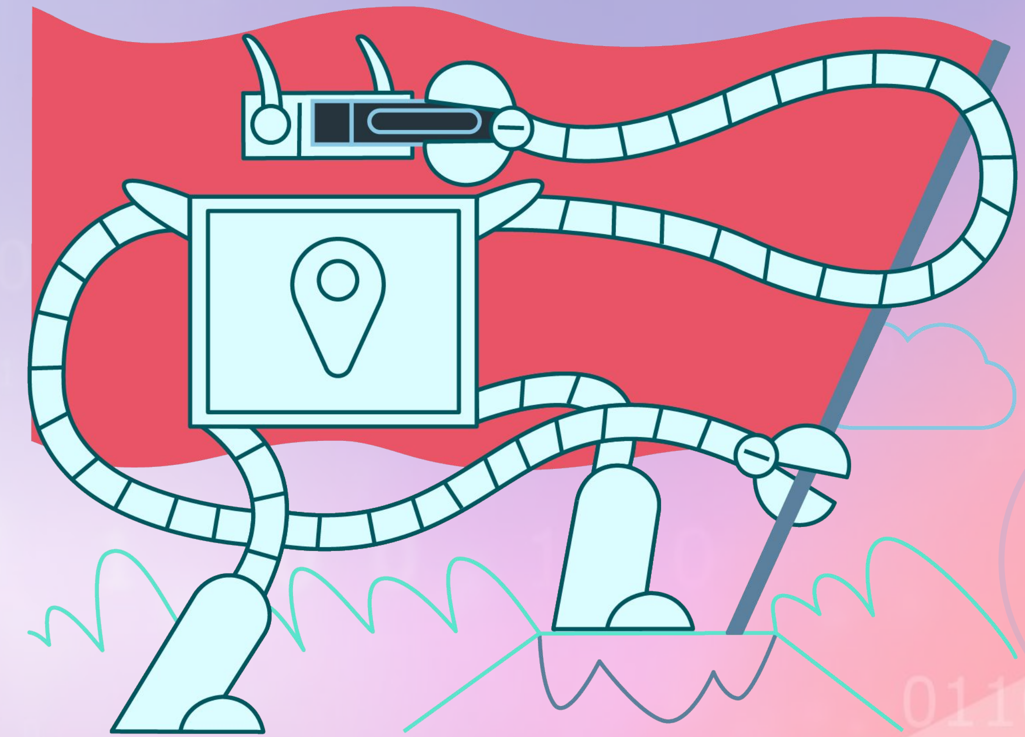
Readable + Modular

Code that is easier to

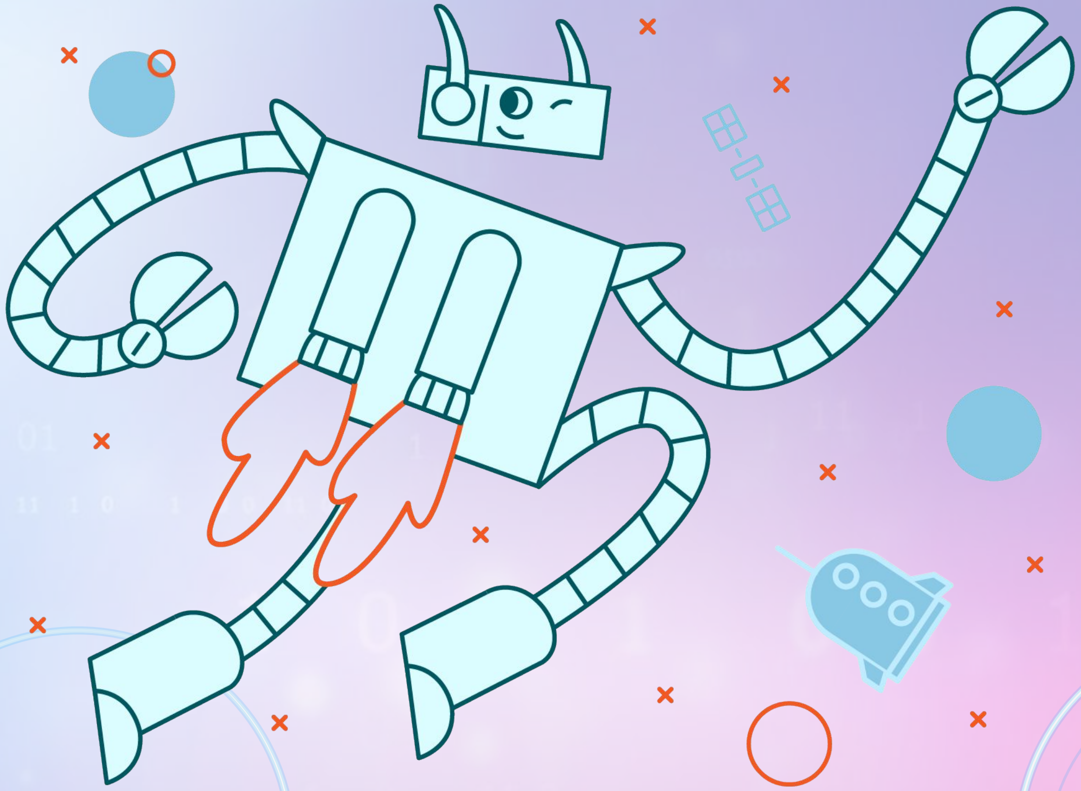
Extend + Debug

Abstract Bake

Take a few minutes to complete the "Abstract Bake" exercise



Solution



JSFiddle

Solution

```
function mixDough(amountFlour, amountWater) {
  console.log(`Mixing ${amountFlour}g flour and ${amountWater}ml water in to dough`)
  return "dough"
}

function bake(bakeTime, dough) {
  console.log(`The bread has been baked for ${bakeTime} minutes!`)
  if(dough == "dough"){
    return "bread"
  }
}

function bakeBread(amountFlour, amountWater, bakeTime){
  let mixture = mixDough(amountFlour,amountWater)
  return bake(bakeTime, mixture)
}

console.log(bakeBread(150, 50, 25))
```

DRY Principle

Don’t Repeat Yourself!

Reduce repetition in code by implementing abstractions! This will avoid redundancy!

Summary

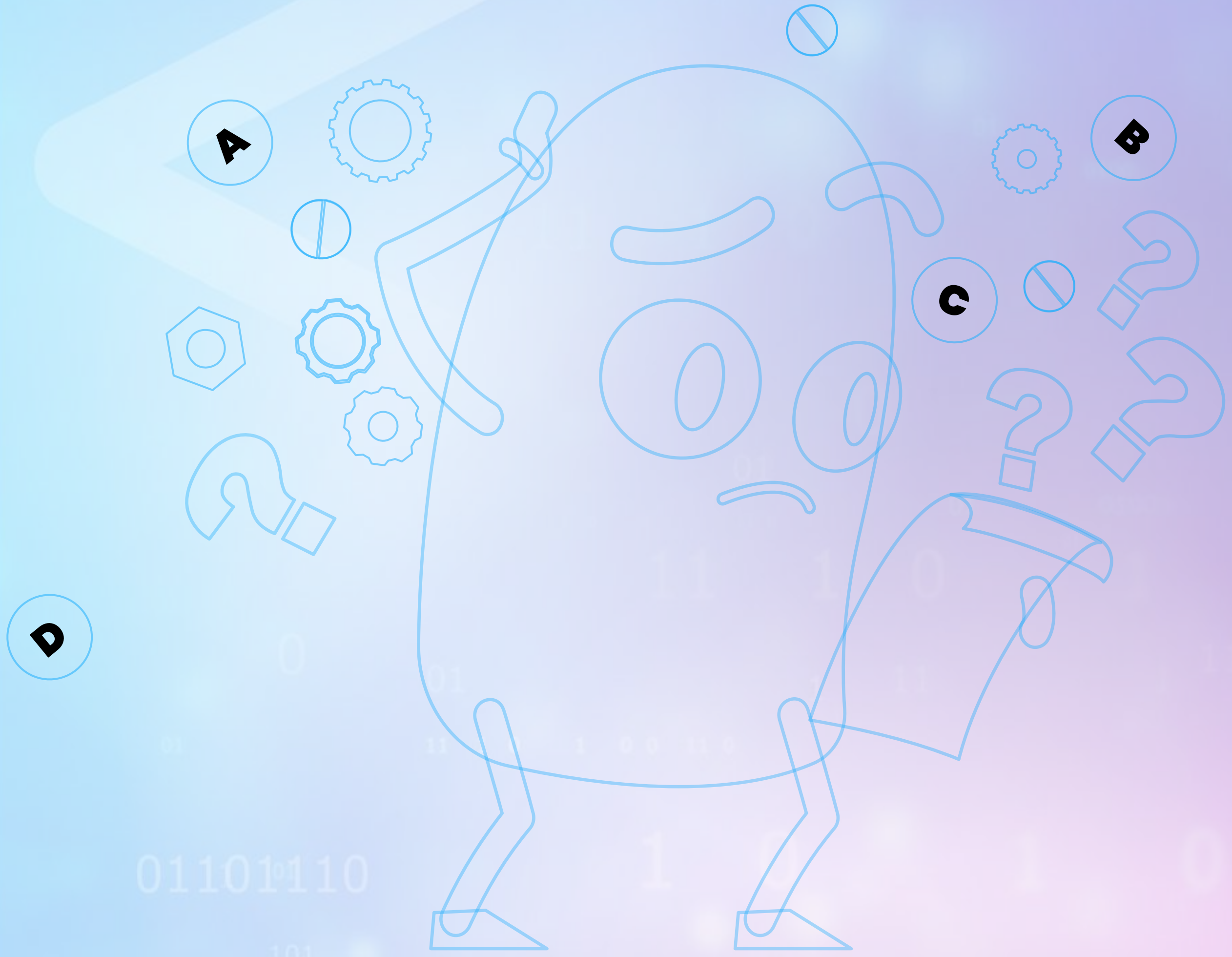
Summary

```
function add(lhs, rhs) {
    return lhs + rhs
}

let result = add(1014, 323)
console.log(result)
```

Parameters: Variables which are provided as inputs in a function

Return the result

Arguments: Values passed into the function as parameters



Questions?

Sentinel

Your Turn!

> Play around, have fun, ask questions!